if0 Language with Booleans

AST · Calculator · Parser · Runner

1. AST

The Exp type represents all possible expressions. Constructors are suffixed with E to distinguish expressions from values.

```
(define-type Exp
  [numE  (n     : Number)]
  [boolE (b     : Boolean)]
  [plusE (left  : Exp) (right : Exp)]
  [cndE  (test  : Exp) (then  : Exp) (else : Exp)])
```

2. Value

The Value type represents what the calculator returns. Constructors are suffixed with V.

```
(define-type Value
  [numV  (n : Number)]
  [boolV (b : Boolean)])
```

3. Helper Functions

3.1 add

Enforces that both operands of + must be numbers.

```
(define (add v1 v2)
  (type-case Value v1
    [(numV n1)
     (type-case Value v2
       [(numV n2) (numV (+ n1 n2))]
       [else (error '+ "expects RHS to be a number")])]
    [else (error '+ "expects LHS to be a number")]))
```

3.2 boolean-decision

Enforces that the condition of if must evaluate to a boolean.

```
(define (boolean-decision v)
  (type-case Value v
    [(boolV b) b]
    [else (error 'if "expects conditional to evaluate to a boolean")]))
```

4. Calculator (Exp -> Value)

Pattern-matches on the AST and returns a Value. Only one branch of cndE is ever evaluated.

```
(define (calc e)
  (type-case Exp e
    [(numE n)      (numV n)]
    [(boolE b)     (boolV b)]
    [(plusE l r)   (add (calc l) (calc r))]
    [(cndE c t e) (if (boolean-decision (calc c))
                      (calc t)
                      (calc e))]))
```

5. Parser (S-Exp -> Exp)

Converts surface syntax into an Exp. Concrete syntax: numbers, #true/#false, {+ e e}, {if e e e}.

```
(define (parse s)
  (cond
    [(s-exp-number? s)
     (numE (s-exp->number s))]
    [(s-exp-boolean? s)
     (boolE (s-exp->boolean s))]
    [(s-exp-list? s)
     (let ([l (s-exp->list s)])
       (cond
         [(symbol=? '+ (s-exp->symbol (first l)))
          (plusE (parse (second l))
                 (parse (third  l)))]
         [(symbol=? 'if (s-exp->symbol (first l)))
          (cndE  (parse (second l))
                 (parse (third  l))
                 (parse (fourth l)))]
         [else
          (error 'parse "unknown operator")]))]
    [else
     (error 'parse "not a number, boolean, or list")]))
```

6. Runner (S-Exp -> Value)

Composes parse and calc into a single pipeline entry point.

```
(define (run s)
  (calc (parse s)))
```

7. Tests

```
(print-only-errors #true)
```

```
; --- numbers ---
(test (run `1)              (numV 1))
(test (run `2.3)            (numV 2.3))
(test (run `{+ 1 2})        (numV 3))
(test (run `{+ {+ 1 2} 3})  (numV 6))

; --- booleans ---
(test (run `#true)          (boolV #true))
(test (run `#false)         (boolV #false))

; --- if with boolean condition ---
(test (run `{if #true  1 2})  (numV 1))   ; true  → then
(test (run `{if #false 1 2})  (numV 2))   ; false → else

; --- nested ---
(test (run `{if #true {+ 1 2} 99})   (numV 3))
(test (run `{if #false 99 {+ 3 4}})  (numV 7))

; --- errors ---
(test/exn (run `{+ #true 1})   "LHS")
(test/exn (run `{+ 1 #false})  "RHS")
(test/exn (run `{if 0 1 2})    "boolean")
```